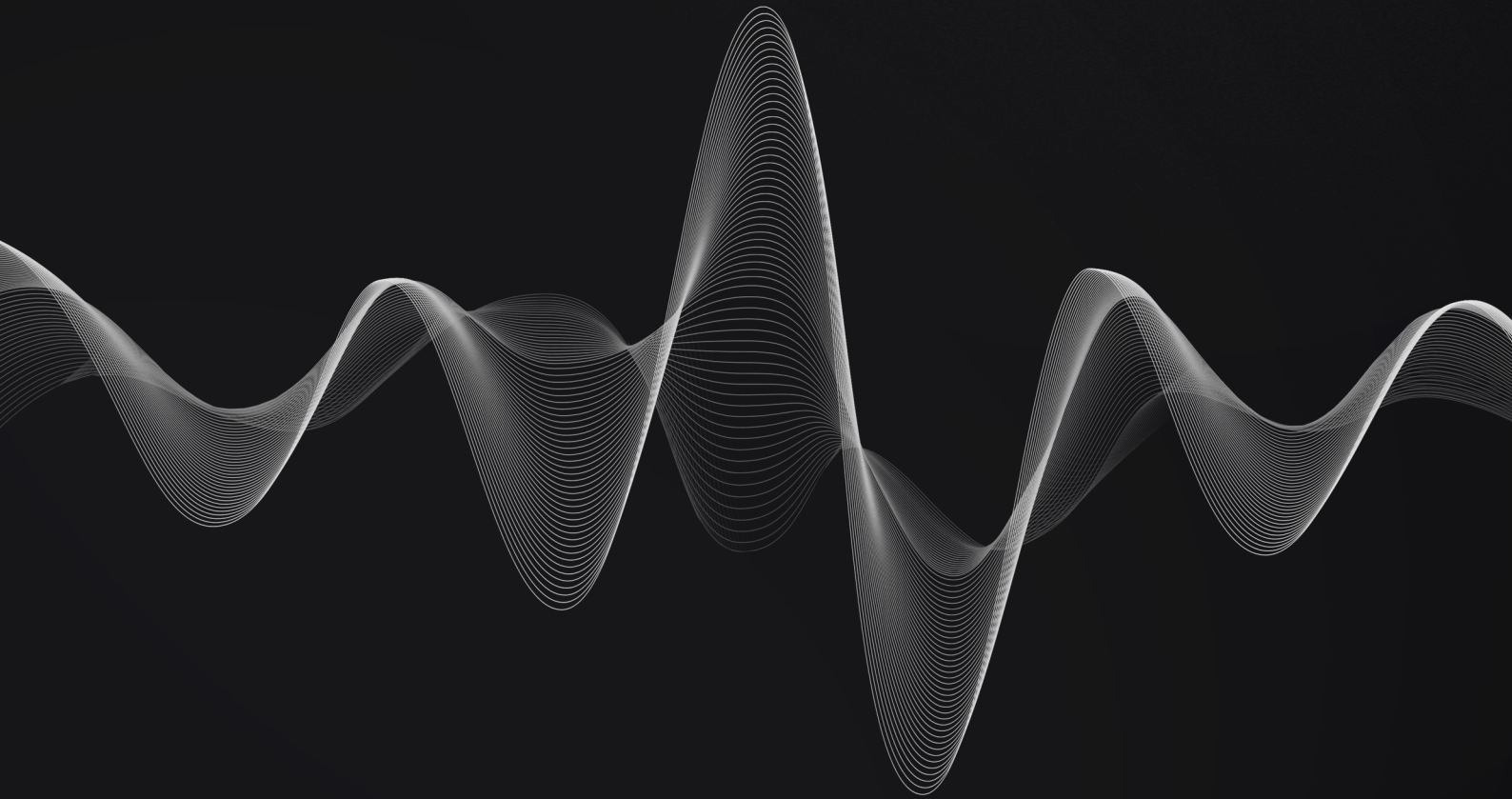


Nodle

Nodle ZKsync Bridge Smart Contract Audit








Document Control

PUBLIC

FINAL(v2.1)

Audit_Report_NODL-BRD_FINAL_21

Sep 11, 2025		v0.1	Luis Arroyo: Initial draft
Sep 11, 2025		v0.2	Luis Arroyo: Added findings
Sep 11, 2025		v1.0	Charles Dray: Approved
Sep 18, 2025		v1.1	Luis Arroyo: Reviewed findings
Oct 27, 2025		v2.0	Charles Dray: Finalized
Oct 27, 2025		v2.1	Charles Dray: Published

Points of Contact	Micha Benoliel	Nodle	micha@nodle.com
	Charles Dray	Resonance	charles@resonance.security
Testing Team	Luis Arroyo	Resonance	luis.arroyo@resonance.security
	João Simões	Resonance	joao@resonance.security
	Ilan Abitbol	Resonance	ilan@resonance.security

Copyright and Disclaimer

© 2025 Resonance Security, Inc. All rights reserved.

The information in this report is considered confidential and proprietary by Resonance and is licensed to the recipient solely under the terms of the project statement of work. Reproduction or distribution, in whole or in part, is strictly prohibited without the express written permission of Resonance.

All activities performed by Resonance in connection with this project were carried out in accordance with the project statement of work and agreed-upon project plan. It's important to note that security assessments are time-limited and may depend on information provided by the client, its affiliates, or partners. As such, the findings documented in this report should not be considered a comprehensive list of all security issues, flaws, or defects in the target system or codebase.

Furthermore, it is hereby assumed that all of the risks in electing not to remedy the security issues identified henceforth are sole responsibility of the respective client. The acknowledgement and understanding of the risks which may arise due to failure to remedy the described security issues, waives and releases any claims against Resonance, now known or hereafter known, on account of damage or financial loss.

Contents

1 Document Control	2
Copyright and Disclaimer	2
2 Executive Summary	4
System Overview	4
Repository Coverage and Quality.....	4
3 Target	5
4 Methodology	6
Severity Rating.....	7
Repository Coverage and Quality Rating.....	8
5 Findings	9
Unbounded ETH Forwarded to Mailbox	10
Pausing Contracts Block Deposit Finalization	11
Missing Zero Address Validation On L1Nodl.constructor()	12
Missing Zero Address Validation On deposit()	13
Lack Of CEI Pattern Implementation	14
A Proof of Concepts	15

Executive Summary

Nodle contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between September 9, 2025 and September 11, 2025. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 2 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 4 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

The ultimate goal of the audit was to provide Nodle with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.



System Overview

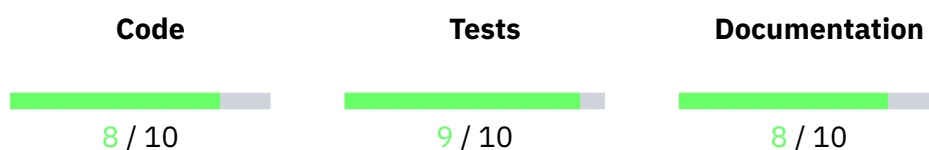
Nodle implemented a ZKsync bridge in order to transfer tokens as they will launch on zkSync Era.

The protocol consists in a L1 endpoint that burns on deposits, verifies L2 proofs, and mints on finalized withdrawals and a L2 endpoint that mints on finalized deposits and burns on withdrawals while sending L2→L1 messages.

The protocol also have its own L1 ERC20 with burn, permit/votes, and role-gated mint for the bridge.



Repository Coverage and Quality



Resonance's testing team has assessed the Code, Tests, and Documentation coverage and quality of the system and achieved the following results:

- The code follows development best practices and makes use of known patterns, standard libraries, and language guides. It is easily readable and uses the latest stable version of relevant components. Overall, **code quality is good**.
- Unit and integration tests are included. The tests cover both technical and functional requirements. Code coverage is 93%. Overall, **tests coverage and quality is excellent**.
- The documentation includes technical details for the code, relevant explanations of workflows and interactions. Overall, **documentation coverage and quality is good**.

Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Repository: [NodleCode/rollup](#)
- Hash: 4668233a2e67e03537a218412adee34d4784d934

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process
- Financial related attacks

Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

Source Code Review - Solidity EVM

During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Solidity EVM audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Reentrancy attacks
- Frontrunning attacks
- Unsafe external calls
- Unsafe third party integrations
- Denial of service
- Access control issues

- Inaccurate business logic implementations
- Incorrect gas usage
- Arithmetic issues
- Unsafe callbacks
- Timestamp dependence
- Mishandled panics, errors and exceptions

Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss
2. Medium - Temporary or partial damage or loss
3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions
2. Likely - Requires technical knowledge or no special conditions
3. Very Likely - Requires trivial knowledge or effort or no conditions

		Likelihood		
		Very Likely	Likely	Unlikely
Impact	Strong	Critical	High	Medium
	Medium	High	Medium	Low
	Weak	Medium	Low	Info



Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.
- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.
- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.

Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance’s Testing Team. This metric characterizes findings as follows:

- **"Quick Win"** Requires little work for a high impact on risk reduction.
-| | **"Standard Fix"** Requires an average amount of work to fully reduce the risk.
- ...| | | **"Heavy Project"** Requires extensive work for a low impact on risk reduction.

RES-01	Unbounded ETH Forwarded to Mailbox	Resolved
RES-02	Pausing Contracts Block Deposit Finalization	Acknowledged
RES-03	Missing Zero Address Validation On L1Nodl.constructor()	Resolved
RES-04	Missing Zero Address Validation On deposit()	Resolved
RES-05	Lack Of CEI Pattern Implementation	Resolved



Unbounded ETH Forwarded to Mailbox

Medium

RES-NODL-BRD01

Data Validation

Resolved

Code Section

- `L1Bridge.sol#L133`

Description

The `deposit()` function forwards the entire `msg.value` to `L1_MAILBOX.requestL2Transaction()` without checking or capping against the required base cost. The `IMailbox` interface shows that `IMailbox.l2TransactionBaseCost()` can be used to provide a bounded amount of Eth as gas.

Recommendation

It is recommended to compute required base cost to cap ETH sent to Mailbox.

Status

The issue has been fixed in `8a99a8e3e1bf4e599b1d751d6a173c2e13650053`.



Pausing Contracts Block Deposit Finalization

Medium RES-NODL-BRD02

Business Logic

Acknowledged

Code Section

- [L1Bridge.sol#L171](#)
- [L1Bridge.sol#L202](#)
- [L2Bridge.sol#L107](#)
- [L2Bridge.sol#L124](#)

Description

The `finalizeDeposit()` function from `L2Bridge` implements `whenNotPaused` modifier. If the contract is paused while L1 has already burned/locked funds, L2 minting is blocked and funds are stuck until unpaused. This functionality might be intended, but often bridges allow finalization while still pausing user-initiated withdrawals.


Something similar happens with `L1Bridge`. When the contracts are paused, the owner can unintentionally lock funds in-flight, as deposit refunds can't be claimed or L2 -> L1 withdrawals can't be finalized.

Recommendation

Consider allowing deposit finalization while paused unless this is the expected functionality

Status

The issue was acknowledged by Nodle's team. The development team stated "We consider the current functionality which pauses all major interactions with the bridge as the intended behavior."



Missing Zero Address Validation On L1Nodl.constructor()

Low

RES-NODL-BRD03

Data Validation

Resolved

Code Section

- [L1Nodl.sol#L15](#)

Description

Input parameter in `constructor()` function is not being validated against the Zero Address.

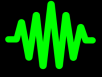
Mistakes can be made by the owner of the smart contract, allowing for unwanted transactions to take place in the future.

Recommendation

It is recommended to perform a validation against the Zero Address to ensure proper variable values are handled properly and successfully.

Status

The issue has been fixed in [ddd2bf327db662ecb6de97bfa86cbaa808855d23](#).



Missing Zero Address Validation On deposit()

Low

RES-NODL-BRD04

Data Validation

Resolved

Code Section

- [L1Bridge.sol#L117](#)

Description

Input parameters in `deposit()` function is not being validated against the Zero Address.

Mistakes can be made by the owner of the smart contract, allowing for unwanted transactions to take place in the future.

Recommendation

It is recommended to perform a validation against the Zero Address to ensure proper variable values are handled properly and successfully.

Status

The issue has been fixed in [87386b59a71b5fef30e96f713b534af433de049](#).



Lack Of CEI Pattern Implementation

Info

RES-NODL-BRD05

Code Quality

Resolved

Code Section

- [L1Bridge.sol#L206](#)

Description

The Check-Effects-Interactions (CEI) pattern is a best practice in Solidity for writing secure smart contracts. It involves structuring the contract's logic in a specific order: checking conditions, making state changes (effects), and then interacting with external contracts.

Recommendation

It is recommended to implement the CEI pattern whenever possible. In this case, `isWithdrawalFinalized` should be marked as `true` after the check of the `proveL2MessageInclusion()` call.

Status

The issue has been fixed in `d11968816e6d2bf70f73821484013ce202029219`.

Proof of Concepts

No Proof-of-Concept was deemed relevant to describe findings in this engagement.